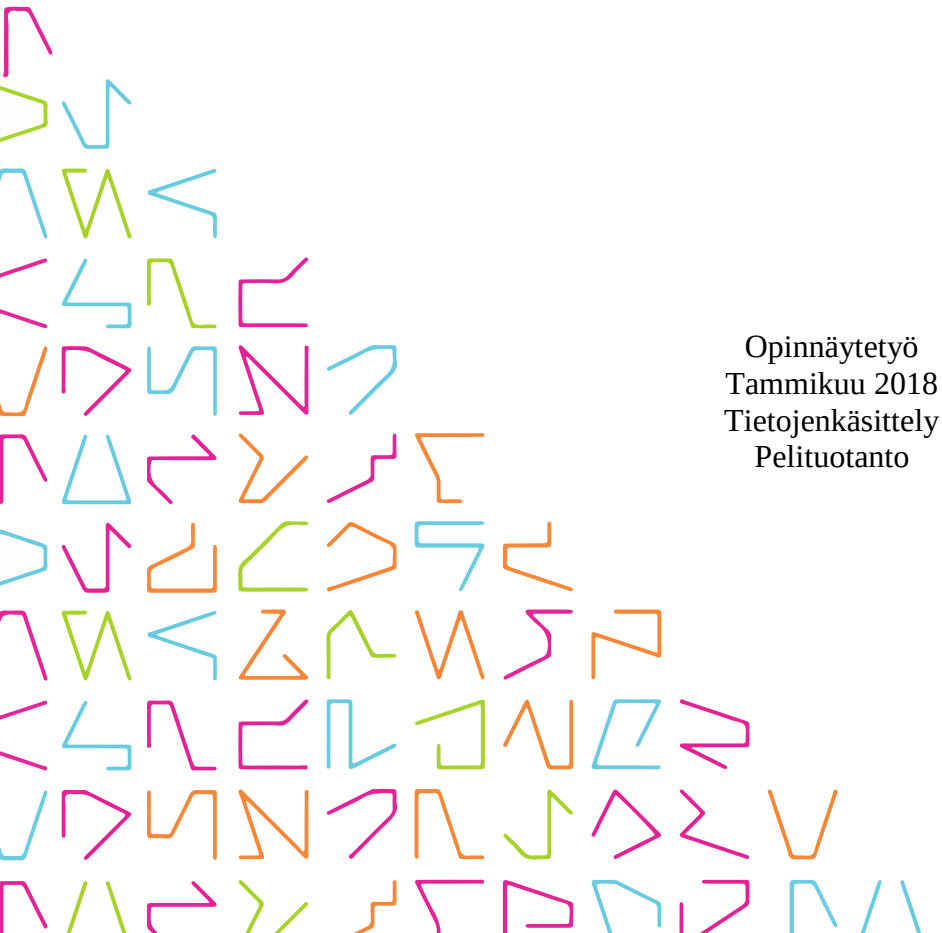
TAMPEREEN
AMMATTIKORKEAKOULU

TEKOÄLYN TOTEUTUS PELIPROJEKTISSA

Tapaus Golden Roll

Kalle Värälä



Opinnäytetyö
Tammikuu 2018
Tietojenkäsittely
Pelituotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Pelituotanto

VÄRÄLÄ, KALLE:
Tekoälyn toteutus peliprojektissa
Tapaus Golden Roll

Opinnäytetyö 38 sivua, joista liitteitä 4 sivua
Tammikuu 2018

Tämän opinnäytetyön aiheena ovat erilaiset tekoälyt peleissä sekä tekoälyn toteuttaminen yhteen mobiilipeliin. Opinnäytetyö toteutettiin tietojenkäsittelyn tradenomin tutkinnon osana pelituotannon suuntautumispolulla. Opinnäytetyön aihealueita ovat pelisuunnittelu ja peliohjelmointi, tarkemmin tekoälyn toteuttaminen.

Tekoälyt valittiin opinnäytetyön aiheeksi, koska ne tekevät pelaamisesta paljon monipuolisempaa. Lähes jokainen ohjelmoija saa joskus tehtäväkseen toteuttaa peliprojektissa jonkinlainen tekoälyn, joten tekoälyyn liittyvän osaamisen kehittäminen on sekä tärkeää että luontevaa.

Opinnäytetyön ensimmäisessä osassa tutkittiin erilaisia tapoja toteuttaa tekoäly ja selvitettiin, millaisia ovat yleisimmät käytetyt tekoälyrakenteet hyödyntäen lähdekirjallisuutta ja pelejä. Toisessa osassa toteutettiin tekoäly Golden Roll -nimiseen peliprojektiin ja tekoälysuunnitelma Caverna–nimiseen peliprojektiin.

Tekoälyrakenteita on monenlaisia ja se, millainen tekoälyrakenne kannattaa valita, riippuu pelin genrestä, pelin sisällöstä, resursseista ja kehittäjien preferensseistä. Golden Rolliin valittiin sääntöpohjainen tekoälyllä generoitu vuorolista, jota skriptattu tekoäly pelaa, koska sillä saatiin luotettavasti odotettuja tuloksia. Lisäksi sen toteuttaminen sopi käytettävissä olleisiin resursseihin. Tekoälyllä saatiin lisäsisältöä Golden Rolliin ja lopputulos oli onnistunut.

Vaikka opinnäytetyön palautushetkellä Golden Roll odottaa vielä julkaisuaan, saa opinnäytetyöstä tapausesimerkin avulla kuvan siitä, millaista tekoälyn suunnittelu ja toteuttaminen mobiilipeliin on. Lisäksi työstä saa pohjatiedot erilaisista tekoälytyypeistä ja lähdekirjallisuutta hyödyntäen voi kiinnostavista rakenteista hakea lisää tietoa.

Asiasanat: tekoäly, pelit, pelituotanto

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Game Development

VÄRÄLÄ, KALLE:
Implementation of Artificial Intelligence in a Game Project
Case Golden Roll

Bachelor's thesis 38 pages, appendices 4 pages
January 2018

The subjects of this thesis are the study of artificial intelligence types in games and implementation of an artificial intelligence in a mobile game.

Artificial intelligences were chosen as the subject because as they arguably make gaming much more interesting. The author had also been given the task of implementing an artificial intelligence in multiple different projects, which further increased the author's motivation to write his thesis on the subject.

In the first part of this study, different ways of implementing artificial intelligence and most popular artificial intelligence structures were researched using source literature and games. In the second part an artificial intelligence was implemented in a game project called Golden Roll and an artificial intelligence design was created for a game project called Caverna.

There are many different artificial intelligence structures, which are generally chosen according to the genre or content of the game, the resources available and the preferences of the developers. The structure used in the Golden Roll project involves a scripted artificial intelligence which uses a turn list generated by a rule-based AI to play the game, because it provides dependable results and realizing it was within the resource limits.

From the thesis one can, through the case study, get a picture of what it is to design and implement a simple artificial intelligence in a mobile game project. One also acquires basic knowledge of different types of artificial intelligence.

Key words: artificial intelligence, game development, games

SISÄLLYS

LYHENTEET JA TERMIT

algoritmi (tässä työssä)	tehtävän koneellinen suoritusohje
backward planning	takaperin suunnittelu
forward planning	etuperin suunnittelu
FSM	äärellinen tilakone
FuSM	sumea tilakone
GOAP	tavoiteorientuituneet toimintasuunnittelijat
hack’n’slash	lähitaistelukeskeinen pelityyppi
HFSM	hierarkkinen äärellinen tilakone
HTN	hierarkkiset tehtäväverkostot
hybridi	usean eri asian yhdistelmä
hyödynlaskentajärjestelmä	agentin halua/tarvetta mittaava tekoälytyyppi
käyttäytymispuu	puumainen tietorakenne tekoälyn toteuttamiseen
shoot-em-up, shmup	räiskintään keskittyvä pelityyppi
soft launch	pelin rajattu julkaiseminen
tekoäly	tietokoneen älykkäästä toiminnasta vastaava komponentti
tilakone	erilaisia mahdollisia tiloja sisältävä järjestelmä
älykäs agentti	tekoälyn ohjaama ohjelmiston pala

1 JOHDANTO

Tämä opinnäytetyö on suunnattu pelin tekoälyn toteuttamista harkitseville, mutta myös peleistä ja niiden tekoälyistä muuten vain kiinnostuneille. Muusta syystä aiheesta kiinnostuneille opinnäytetyö antaa uusia näkökulmia pelaamiseen ja peleihin. Kuitenkin tietynlainen tekninen kiinnostus pelien toteuttamiseen ja pohjatiedot ohjelmoinnista helpottavat joidenkin opinnäytetyön osioiden seuraamista.

Tekoäly valikoitui opinnäytetyön aiheeksi Greener Grass -peliyhtiön Golden Roll -peliprojektin kautta. Opinnäytetyöprosessin aloittamisen aikaan olin Greener Grassilla töissä harjoittelijana ja sain Golden Roll -pelin toteuttamisessa paljon vastuuta. Tekoälyn luominen oli jo ensimmäisiä Golden Roll -projektissa saamiani tehtäviä, koska pelille oli luotava tietyllä tavalla pelaava tekoäly. Sen suunnittelu- ja toteutusvaiheista kerrotaan tässä opinnäytetyössä myöhemmin omassa luvussaan.

Tekoälyn toteutettuani huomasin toteuttaneeni lähes jokaisessa aiemmassakin peliprojektissani tekoälyjä eri tavoilla, mikä kasvatti innostustani tekoälyjä ja yleisesti aihepiiriä kohtaan. Myös Golden Rollin tekoälyä kehitetään edelleen pelin jäätyä yhtiön pidempiaikaiseksi projektiksi.

Muita pelejä, joita työssäni käsittelen, ovat DroneCatcher, Whirlsteel, Just Cause 3, theHunter: Call of the Wild ja Caverna. DroneCatcher ja Whirlsteel ovat kouluprojekteja, joihin molempiin olen itse toteuttanut tekoälyn. Just Cause 3 ja theHunter ovat Avalanche Studios -peliyhtiön pelejä, joissa on käytetty erinomaisesti käyttäytymispuu-tekoälyrakennetta. Caverna on oma peliprojektini, jota olen tehnyt vapaa-ajallani. Tarkoitus oli toteuttaa Cavernaan tekoäly osana opinnäytetyötä, mutta opinnäytetyö rajautui myöhemmin pienemmäksi. Cavernan tekoälyn rakennesuunnitelmia löytyy kuitenkin myös liitteistä (liite 1), jotta lukija voi asiasta kiinnostuessaan tutustua myös niihin.

Tekoälyjen eri muotoja käsitellään opinnäytteessä luvussa 3 Tekoälyn rakenne, mutta yleisesti jo tässä kohtaa voi todeta, että tekoälyn tuoma tuki ja vastus ovat erittäin tärkeitä ja olennaisia suurimmassa osassa olemassa olevista peleistä ja että tekoälyn uudet ja yhä monipuolisemmat muodot ovat myös tulevaisuuden pelialan kehityskohde. Tekoälyn toteuttamiseen käytetyt tavat ovat kehittyneet 80-luvun skriptatuista tekoälyistä nykyisiksi oppiviksi ja tavoitteellisiksi tekoälyiksi.

Erilaisissa peliprojekteissa on erilaiset tekoälytarpeet, joten tekoälyn kehittämistä aloittaessa on hyvä tietää, minkälaisia erilaisia vaihtoehtoja muissa projekteissa on käytetty. Siksi tässä opinnäytetyössäkin pyritään antamaan monipuolisesti esimerkkejä erilaisista tekoälyn toteutuksista. Aivan kaikkia tekoälyrakenteita, kuten reitinhakualgoritmeja, ei opinnäytetyössä voida esitellä aiheen laajuuden takia. Vaikka reitinhaku onkin suuri tekoälyn osa-alue, on se jätetty tarkoituksella tästä opinnäytetyöstä pois.

Tämän opinnäytetyön tarkoituksena on helpottaa tekoälyrakenteen valintaa, sekä antaa esimerkkejä siitä, miten rakenteita on käytetty muissa projekteissa.

2 TEKOÄLY PELEISSÄ

Tekoäly on ollut medioissa pinnalla eri konteksteissa kasvavissa määrin, mikä havainnollistuu opinnäytetyössä myöhemmin annettavissa esimerkkeissä. Tekoäly on ohjelmistollinen komponentti, joka ohjaa tietokoneen älykästä (älykkäältä vaikuttavaa, ihmismäistä ajattelua muistuttavaa) toimintaa. Konsoli- ja tietokonepeleissä tekoäly on ollut oleellinen osa parantamassa pelikokemusta siitä asti, kun yksin pelattavat pelit tulivat markkinoille.

Tekoäly voi olla hyvin yksinkertainen tai äärimmäisen monimutkainen. Erilaisia tekoälyn sovelluksia on käytössä kielten kääntämisestä ja tuottamisesta aina kuvanlukuun ja muodontunnistukseen. Tekoälyä käytetään tehtaissa ja robottitekniikassa sekä erilaisissa viihdesovelluksissa. Muun muassa metroaseman automaattiportti on esimerkki arkisesta tekoälystä: portissa tekoäly lukee syötetyn metrolipun erilaisilla sensoreilla, kuten magneetti- tai RFID-järjestelmillä, ja hyväksytyn lipun luettuaan asettaa tilakoneen tilaksi “portti auki”, jolloin asiakas voi jatkaa eteenpäin. Kun asiakas on kulkenut portin läpi, asetetaan tilakoneen tilaksi “portti kiinni”. Kaikki tämä on hyvin yksinkertaisen tekoälyn toimintaa.

Peleissä tekoäly vastaa pelin hahmojen ja osien sellaisista toiminnoista, jotka eivät ole pelaajan määräämiä. Tekoälyn avulla peleihin saadaan huomattavasti uusia ominaisuuksia ja ulottuvuutta. *Älykäs agentti* (Intelligent agent) on pelin palanen, joka toimii tekoälyn ohjaamana. Tietokoneellinen tekoäly ja sitä kautta myös pelien tekoäly onkin älykkäiden agenttien kehittämistä (Russel & Norvig 2010, viii; Poole, Mackworth & Goebel 1998, 1). Näitä agentteja ohjaamalla saadaan peliin orgaanisuuden tuntu, tai ylipäätään sisältö. Ilman tekoälyä peli ei välttämättä olisi yhtä puoleensavetävä.

Pelin puoleensavetävyyden kannalta haastetta tuottavissa tekoälyissä tärkeää on pitää tekoäly sopivan vaikeana, mutta kuitenkin pelaamisen ja eteenpäin menemisen kannalta mahdollisena. Yleisenä sääntönä pidetään, että tekoäly ei saa viedä pelistä hauskuutta, mutta että se saa olla kuitenkin hiukan “liian vaikea”. Esimerkiksi pelaajan onnistuminen ei ole välttämättä niin tärkeää kuin se, että pelaajalla on pelatessa hauskaa (Dill 2013, 4).

Tekoälyn kannalta olennaista on, että pelaajat haluavat osallistua heille luotuun elämykseen. Yleensä pelaajien on vaikea pitää uskottavina selkeän keinotekoisia hahmoja ja ympäristöjä. Pelintekijän vastuulla on luoda tarpeeksi voimakas ja mukaansatempaava illuusio, jotta pelaajat pystyvät luottamaan pelin vaihtoehtoiseen todellisuuteen ja sulkemaan pois luonnollisen epäuskoisuutensa pelin epänormaaleista tai epäluonnollisista tapahtumista. Pelintekijän ja tekoälyn tavoitteena on siis pitää yllä pelaajan "suspension of disbeliefiä" eli eläytymistä pelin fiktiiviseen sisältöön. (Dill 2013, 4.)

Heikko tekoäly viittaa koneeseen, jolla ei ole kykyä käsittää ihmisen kognitiivista kyvykkyyttä koko laajuudessaan, eikä heikko tekoäly myöskään saavuta omaa konetietoisuuttaan. *Vahva tekoäly* tarkoittaa puolestaan konetta, joka lähestyy inhimillisen älykkyyden tasoa tai jopa ylittää sen. Käytännössä kaikki nykyisten pelien ja ohjelmien tekoälyt ovat heikkoja tekoälyjä. Heikko tekoäly voi olla hyvinkin älykkään oloinen, mutta siltä puuttuu tietoisuus ja todellinen oppimiskyky.

Tavanomaisesti tekoälystä puhuttaessa ymmärretään tekoälyn olevan science fictionistakin tuttu vahva tekoäly. Vahva tekoäly oppii ympäristöstään ja tekemisistään aidosti älykkäästi, osaten hyödyntää oppimaansa ja kehittyen siten jatkuvasti. Vahva tekoäly omaa tavallaan oman tietoisuuden, joten kyseessä on eräänlainen teoreettinen, ohjelmistollinen organismi.

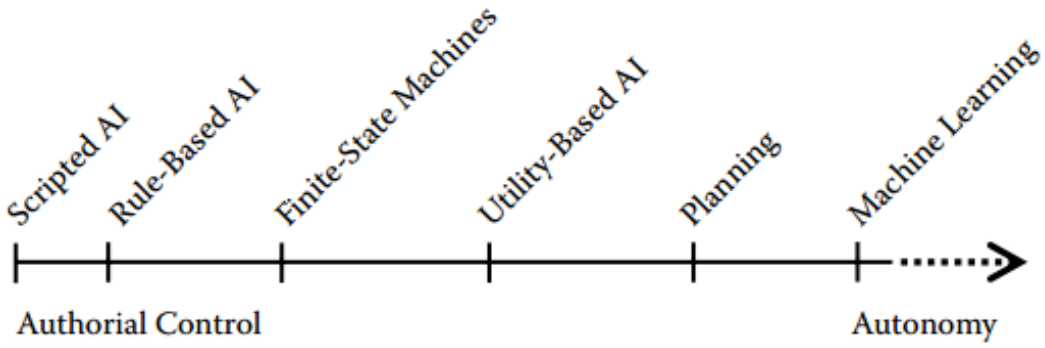
Vahvan tekoälyn hyötynä on sen älykkyys ja tehokkuus. Oppimiskyvyn ja itsekehittämisen vuoksi vahvaan tekoälyyn liittyy kuitenkin mahdollisuus siitä, että tekoäly joskus ohittaisi ihmisen orgaanisen älykkyyden. Vahva tekoäly on kuitenkin toistaiseksi spekulaatiota, joten on syytä toistamiseen tarkentaa, ettei pelinkehityksessä ole kyse itseään kehittävästä vahvasta tekoälystä, vaan tekoäly toimii sille annettujen ohjeiden puitteissa.

3 TEKOÄLYN RAKENNE

Tekoälyn toteutustapoja ja rakenteita on useita. Tekoälyn tyyppi ja toteutustapa riippuu pelin genrestä ja sisällöstä sekä siitä, mitä tekoälyltä halutaan.

Laajemmat ja monimutkaisemmat pelit tarvitsevat useita erilaisia tekoälyjä. Esimerkiksi reaaliaikaisissa strategiapeleissä vastustajan joukkoja ja rakennuksia ohjaa yksi suuri, vastapelaajaa simuloiva tekoäly. Tämän lisäksi joukkojen yksiköillä on omat tekoälynsä, jotka ohjaavat kyseisten yksiköiden toimintaa. Lisäksi tekoäly saattaa rakenteena toteuttaa useita erilaisia tekoälytyyppejä sisäkkäin. Esimerkiksi *tilakoneen* tilat on usein toteutettu sääntöpohjaisena tai skriptattuna tekoälynä, joista kerrotaan lisää omissa luvuissaan.

Kuviossa 1 näkyy janalla erilaisten tekoälytyyppien autonomisuusaste skriptatusta tekoälystä kohti vahvaa tekoälyä. Strategiapelin vastapelaajaa simuloiva tekoäly sijoittuu kaaviossa oikeaan reunaan, Planning-kohdan tietämille. Joukkojen yksiköiden omat tekoälyt taas sijoittuvat lähemmäs vasenta reunaa Finite-State Machines -kohdalle. Kun tekoälyn toteutuksessa siirrytään puhtaasti skriptatusta tekoälystä kohti autonomisemmin toimivaa tekoälyä, lähestytään vahvaa tekoälyä.



KUVIO 1. Autoritäärisen ja reaktiivisen tekoälyn kompromissi (Dill 2013, 6).

Skriptattu-sanalla tarkoitetaan ohjelmaa, joka suorittaa annetut tehtävät ennaltamäärätyssä järjestyksessä. Autonomisempi tekoäly voi suorittaa joitain valintoja itsenäisemmin. Näitä tekoälyn muotoja käsitellään opinnäytetyössä hiukan laajemmin vielä seuraavassa luvussa. Peleissä tasapaino täysin skriptatun ja autonomisemman tekoälyn välillä riippuu suunnittelusta. Suunnitteluun vaikuttavia kysymyksiä ovat esimerkiksi:

Kuinka paljon tekoälyn halutaan vievän resursseja? ja Kuinka älykäs tekoäly on tarpeen?

3.1 Yksinkertainen tekoäly

Skriptattu tekoäly

Kun tekoäly ei ota ympäristöään huomioon ollenkaan, kyseessä on puhtaasti *skriptattu tekoäly* (scripted ai). Tämänkaltainen tekoäly toteuttaa ennalta määrättyä suunnitelmaa, mikä tarkoittaa sitä, että jokaisella pelikerralla tekoäly suorittaa täsmälleen samat asiat. Tämän tekoälyn hyviä puolia on se, että se on täysin ennalta määrättävissä, jolloin bugit on helppo karsia. Huono puoli on se, että pelaaja huomaa nopeasti kaavat, joita tekoäly noudattaa. Skriptatulla tekoälyllä ei siis pystytä toteuttamaan kovin orgaanista maailmaa. Nykypeleissä tämänkaltaisia tekoälyjä käytetään lähinnä tarinankerrontaan tai osana laajempaa tekoälyä.

Dronecatcher (https://www.kongregate.com/games/ToPeKaMaArSa/dronecatcher) on TAMKilla opiskelijatyönä toteutettu ylhäältäpäin kuvattu *shoot-em-up* -tyyppinen selainpeli. Itse olin vastuussa pelin vihollisista ja niiden käyttäytymisen taustalla olevasta tekoälystä. Vastustajan ilmestyessä ruudulle alkaa tekoäly toteuttaa lennossaan ajastettuja kuvioita ja ampusmiskaavaa, jotka on ennalta määritelty editorissa. Pelissä pystyy opettelemaan vastustajan liikkeet ja toiminnot ulkoa. Peli onkin hyvä esimerkki skriptatun tekoälyn toiminnasta käytännössä.

Sääntöperusteinen tekoäly

Erona skriptattuun tekoälyyn, *sääntöperusteinen* (rule based) *tekoäly* osaa ottaa jollakin tasolla ympäristöään huomioon, esimerkiksi “kultaa on kerätty tarpeeksi” -> “toteuta tämä skripti”. Sääntöperusteinen tekoäly tietyllä tapaa jatkaa skriptattua tekoälyä, antaen tekoälylle informaatiota, jonka varassa toimia. Sääntöperusteisen tekoälyn hyvät ja huonot puolet ovat hyvin samanlaiset kuin skriptatun tekoälyn. Uutena hyvänä puolena sääntöperusteisessa tekoälyssä verrattuna skriptattuun on se, että tekoälyn näennäinen älykkyys vähentää ennalta-arvattavuutta. Olen itse käyttänyt myös sääntöperustaista tekoälyä opiskeluprojektissa TAMKilla.

Whirlsteel on TAMKilla aloitettu *hack-n-slash* -tyyppinen pc-peli peliohjaimelle. Samoin kuin DroneCatcheriin, loin myös Whirlsteeliin viholliset ja niiden tekoälyn. Suurimpana erona DroneCatcherin vihollistekoälyn toimintaan on se, että vaikka Whirlsteel-pelissä tekoälyn toiminta on ennalta arvattavissa, viholliset tekevät joitain asioita hieman järkevämmin, koska tekoälylle on ohjelmoitu tiettyjä sääntöjä. Jokaisella vihollisella on tietynlainen aistiympyrä, minkä sisällä ne näkevät pelaajan. Kun pelaaja on tämän ympyrän sisällä, osaa vihollinen seurata, hyökätä ja väistää pelaajaa.

3.2 Tilakoneet

Äärellinen tilakone

Äärellinen tilakone (äärellinen automaatti, Finite-state machine, FSM) on esimerkiksi Fun & Houletten (2004, 283), Schwabin (2004, 241) sekä Dawen ym. (2013, 48) mukaan tällä hetkellä yleisin käyttäytymisen mallintamisalgoritmi pelien tekoälyille. Rakenteellisesti yksinkertainen ja helposti koodattava tilakone on helposti testattava, intuitiivinen, monipuolinen ja tehokas tekoälyrakenne (Buckland 2005, 43).

Eräs tilakoneen ominaisuuksista on, että se on helppo muuttaa graafiseksi esitykseksi. Äärellinen tilakone jakaa tekoälyn kokonaisuuden pienemmiksi osiksi, joita kutsutaan tiloiksi. Jokainen tila kuvastaa tietynlaista tekoälyn käyttäytymistä, ja vain yksi tila voi olla kerrallaan aktiivisena. Tilat vaihtuvat siirtymillä, jotka toimivat aina tiettyjen sääntöjen mukaisesti. (Dawe, Gargolinski, Dicken, Humphreys & Mark 2013, 48.)

Hahmottelemassani luolanhallintapeli Cavernassa (kts. myös liite 1) hahmolle voidaan antaa käsky “harjoittele taistelua”. Tämä käsky nostaa tämän tilan hahmon ensimmäiseksi prioriteetiksi, jolloin hahmo lähtee suorittamaan tilan määräämiä sääntöjä. Vihollisen hyökätessä lähistölle hahmo vaihtaa tilansa “taistele”-tilaan.

Äärellisen tilakoneen suurimmat ongelmat syntyvät tilojen määrän kasvaessa. Kun tilojen määrä kasvaa, myös siirtymien määrä kasvaa huimasti. Ohjelmiston rakenne monimutkaistuu, jolloin sitä on hankala esittää graafisesti ja koodin lukeminen vaikeutuu. Uusien tilojen lisääminen sekoittaa logiikkaa huomattavasti.

Mikäli luolanhallintaesimerkin hahmon haluttaisiin jatkavan “taistelu”-tilan jälkeen edellistä toimintoaan, täytyisi taistelutilasta olla siirtymä myös tähän tilaan. Hahmolla voisi tässä vaiheessa olla neljä eri tilaa: “harjoittele”, “nuku”, “syö” ja “taistele”. Erillisiä siirtymiä tilasta toiseen olisi siis yhteensä kaksitoista. Jos mahdollisten tilojen joukkoon halutaan lisätä vaikka “rentoudu”-tilan, se merkitsisi siirtymien kasvua yhteensä kahdeksaantoista.

Toinen äärellisen tilakoneen heikkouksista on sen joustamattomuus. Tilakone voi olla kerrallaan vain yksi tila aktiivisena, joten erilaiset tilojen yhdistelmät eivät ole mahdollisia. Jos designer päättäisi, että hahmo voi harjoitella ja syödä yhtä aikaa, täytyisi tälle luoda äärellisessä tilakoneessa uusi tila sen mahdollistamiseksi.

Hierarkkinen äärellinen tilakone

Hierarkkinen äärellinen tilakone (Hierarchical Finite-State Machine, HFSM) korjaa ongelmia, joita yksinkertaisessa äärellisessä tilakoneessa esiintyy. Tällaisessa tilakoneessa jokainen tila voi olla itsenään oma tilakoneensa. Tällöin logiikka saadaan osioitua selkeämmin hallittavissa oleviin osiin, kuin mitä pelkkä äärellinen tilakone mahdollistaa. (Dawe ym. 2013, 50.)

Jos käyttäisin Caverna-luolanhallintapelissäni hierarkkista äärellistä tilakonetta, voisin siirtää hahmon tilat “nuku” ja “syö” “tarpeet”-tilan alatiloiksi, sekä tilat “harjoittele” ja “rentoudu” “toiminta”-tilan alle (kts. liite 1). Uudenlaisen ryhmittelyn ansiosta siirtymiä on enää kolme, ja myös uusia tiloja voidaan lisätä hierarkkisesti ilman, että siirtymien määrä kasvaa kohtuuttomasti. Eräs hierarkkisen tilakoneen hyvä puoli on myös se, että se muistaa, missä tilassa tilakone oli, kun siitä siirryttiin muuhun tilaan. Se siis osaa jatkaa samasta kohdasta myöhemmin (Dawe ym. 2013, 51). Hierarkkinen tilakone toimii suuressa mittakaavassa paljon sujuvammin kuin pelkkä äärellinen tilakone.

Siispä, kun luolanhallintaesimerkin hahmo on esimerkiksi siirtynyt vihollisen lähestyttyä “taistelu”-tilaan, taistelun jälkeen hahmon on mahdollista jatkaa siitä, mihin se viimeksi jäi. Hahmo oli harjoittelemassa, joten “toiminta”-tilassa on muistissa tämä tila. Taistelusta palatessa hahmo jatkaa harjoittelemista. Samalla tavalla “taistelu”-tilalla on muistissa, mistä tilasta sinne siirryttiin (toiminta), joten siirtymä takaisin tapahtuu automaattisesti taistelun päätyttyä.

Sumea tilakone

Sumea tilakone (Fuzzy-State Machine, FuSM) on teoriassa kuin äärellinen tilakone, mutta boolean-logiikka laajennetaan siinä hyväksymään myös osittaiset totuudet (Schwab 2004, 281). *Sumeaa logiikkaa* voisi kuvata sellaiseksi logiikaksi, jota ihmiset tavallisesti käyttävät päätöksenteossaan. Sumeaa logiikkaa voidaan ajatella olevan esimerkiksi toteamukset “on kylmä”, “juoksen nopeasti” tai “rahaa on paljon”. Näiden toteamusten totuusarvo voi riippua toteamuksen esittäjästä, konteksista tai useasta muusta eri asiasta (Buckland 2005, 415–416).

Schwab (2004, 283) väittää, että sumeat tilakoneet ovat yleistymässä tekoälyn toteutuksessa johtuen äärellisten tilakoneiden ennalta-arvattavuudesta. Sumeat tilakoneet ovat kuin tietynlaisia äärellisia tilakoneita, jotka vain hyväksyvät useamman tilan yhtäaikaisen aktivaation nykyiseksi tilaksi. Tämän lisäksi tiloilla on tietty taso, jolla voi määrätä kuinka aktiivisesti jokin tila on päällä. Tosiasiassa sumeat tilakoneet eivät edes ole tilakoneita, koska järjestelmä ei ole missään vaiheessa yksittäisessä tilassa. Sen sijaan tilakone onkin enemmän sumea tietojärjestelmä, jossa usea eri tarkistus voi olla osittain totta yhtäaikaa. (Schwab 2004, 283–284.)

Nykyään sumeita tilakoneita ei käytetä kovin paljoa, vaan järjestelmän ovat korvanneet erilaiset hyödynlaskentajärjestelmät, tavoiteorientoituneet toimintasuunnittelijat sekä erilaiset hybridit. Sumeiden tilakoneiden hyötynä verrattuna äärelliseen tilakoneeseen on usean tilan yhtäaikainen aktivoituminen, jolloin esimerkiksi luolanhallintapelin hahmo voisi syödä harjoitellessaan. Sumea logiikka on kuitenkin vaikeaa seurata ja varsinkin suurissa ja monimutkaisissa järjestelmissä siitä voi tulla todella sekavaa. Voi olla, että sumeiden tilakoneiden käyttö on vähentynyt juuri siksi, että ne ovat vaikeaselkoisempia ja vaikeammin suunniteltavia kuin monet muut rakenteet.

3.3 Käyttäytymispuu

Käyttäytymispuu (Behaviour Tree) kuvaa tietorakennetta tietystä juurisolmusta (root node) alaspäin ja koostuu käyttäytymisistä, joita agentti voi suorittaa. Jokaisen käyttäytymisen on mahdollista sisältää alenevia käyttäytymismahdollisuuksia, jolloin käyttäytymisten esitys muodostaa puun kaltaisen graafisen esityksen eli käyttäytymispuun.

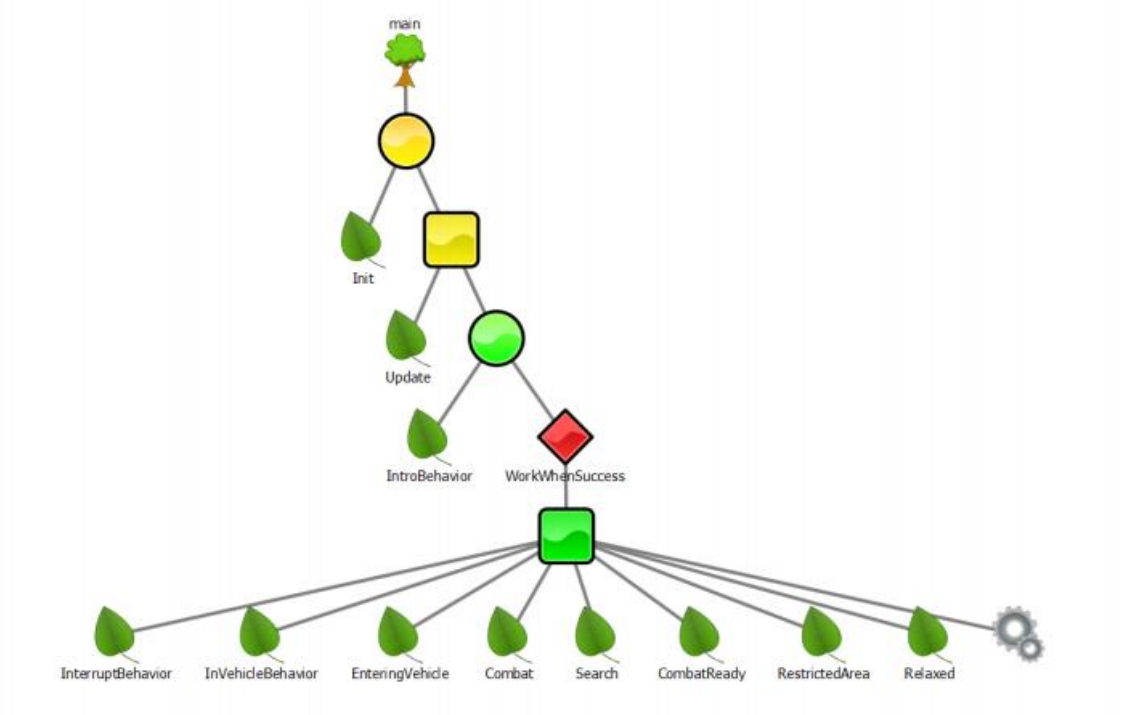
(Dawe ym. 2013, 52.) Yleisesti algoritmi ajaa käyttäytymispuuta kuvion 2 pseudokoodin esittämällä tavalla.

```
aktiivinenSolmu = puu.juuriSolmu;
while (aktiivinenSolmu on olemassa) {
        if (aktiivinenSolmu.Esivaatimus on tosi) {
                ajoLista.lisää(aktiivinenSolmu);
                aktiivinenSolmu = aktiivinenSolmu.Lapsi;
        } else {
                aktiivinenSolmu = puu.Sisar(aktiivinenSolmu);
        }
}
ajoLista.kaikille(solmu => aja(solmu));
```

KUVIO 2. Käyttäytymispuun läpikäyntialgoritmi.

Kuviossa 3 esitetään Avalanche Studios -peliyhtiön Just Cause 3 –räiskintäseikkailupelin sotilaan käyttäytymispuuta. Puun juurisolmusta seuraavalla tasolla on vaihtoehtoina “alustus” tai “seuraava taso”, seuraavalla tasolla taas “päivitys” tai “seuraava taso” ja niin edelleen. Kuviossa on käyttäytymispuun julkinen osa, mutta todellisuudessa puu on monimutkaisempi, esimerkiksi “combat”-solmu jatkuu vielä huomattavasti syvemmälle.

Käyttäytymispuussa jokaisella solmulla on oma ennakkoehtonsa, joka määrittää tilat, joissa agentti suorittaa kyseistä käyttäytymistä. Jokaisella solmulla on myös oma toimintansa, jonka agentti suorittaa toteuttaessaan käyttäytymismallia. Algoritmin suorittaminen aloitetaan aina juuresta siirtyen sitten puussa alaspäin vaadittujen ennakkoehtojen toteutuessa eri solmuissa. Jokaiselta puun tasolta on mahdollista valita vain yksi käyttäytymismalli.



KUVIO 3. Just Cause 3 -pelin vihollisyksikön käyttäytymispuu (Meyer 2017).

Käyttäytymispuun hyvät ja huonot puolet ovat hyvin samankaltaiset kuin hierarkisessa tilakoneessa, mutta koska puuta käydään aina läpi alkaen juurisolmusta, ei solmujen välillä tarvita samanlaisia siirtymiä, kuin tilakoneen tilojen. Tämän takia käyttäytymispuuta usein käytetään monimutkaisempien tekoälyjen toteuttamisessa.

Erinomainen tekoäly - theHunter: Call of the Wild

Avalanche Studios -peliyhtiön pelissä theHunter: Call of the Wild on aivan erinomaisesti toteutettu tekoäly. Useat metsästystä harrastavat pelaajat ovat sanoneet pelin eläinten tekoälyn imitoivan lähes täydellisesti oikeiden metsäneläinten älykkyyttä. Tekoälyn suunnittelija Karine Skoog (2017) kertoo blogissaan ottaneensa projektin vastaan innolla, päästen keskittymään täysin tekoälyn suunnitteluun. Skoogin tehtävänä oli saada tekoäly vastaamaan metsästäjien odotuksia eläinten käytöksestä kuitenkin siten, että samalla tekoäly nivoutuu hyvin pelin muihin toimintoihin.

Tekoälyn suunnittelua helpotti Avalanche Studios -yhtiön omaa sisäisen Apex Engine -pelimoottorin mukana tuleva käyttäytymispuuteknologia ja graafinen suunnittelutyökalu. Tämä vapautti resursseja eläinten käyttäytymisen ja tarpeiden tutkintaan sekä ympäristöjen ja laumojen sisäisen käytöksen suunnitteluun. (Skoog 2017.) Koska kyseessä on

peli, niin realismin hakemisen lomassa täytyy pelistä tehdä mielekästä pelattavaa sekä pelillistää metsästystä.

Vaikka käyttäytymispuu on teoriassa melko yksinkertainen tekoälytyyppi, voidaan sillä saada aikaan erittäin realistisestikin käyttäytyvä tekoäly, kun sitä käytetään työkaluna tarpeeksi monipuolisesti ja kun tekoälyä kehitetään tarpeeksi pitkälle. Suuret puut voivat olla vaikeita lukea, mutta kun ne on hierarkkisesti järjestelty ja käytössä on valmiita graafisia työkaluja, suunnittelutyö ja toteuttaminen helpottuvat huomattavasti. Just Cause 3 ja varsinkin theHunter todistavat, kuinka suurissakin yrityksissä ja peliprojekteissa voidaan hyödyntää "vähemmän älykkäitä" tekoälytyyppejä monipuolisesti ja saada aikaan toimiva näennäinen älykkyys.

3.4 Hyödynlaskentajärjestelmät

Suuri osa tietokoneiden ja tekoälyjen logiikasta toimii puhtaasti boolen logiikan joko-tai -arvoilla, eli on pelkkiä "kyllä vai ei" -kysymyksiä. Myös päätelmät tällaisesta loogisesta järjestelmästä ovat yleensä yhtä polarisoituneita. (Dawe ym. 2013, 53.) Ongelmallista tässä on se, että yleensä päätöksen teko ei ole näin siistiä, vaan päätöstä tehdessä pitää vastata useaan kysymykseen, joissa "kyllä vai ei" -vastaus ei edes riitä, vaan vastaus on hyvin monitahoinen. Esimerkiksi pelissä tilanteen tulkitsemiseen voivat vaikuttaa kuinka hyvässä kunnossa hahmo on, kuinka voimakas vihollinen on, kuinka lähellä muita kavereita on tai mitä tilaefektejä kyseisellä hetkellä on päällä.

Hyödynlaskentajärjestelmät (Utility Systems) vastaavat tähän ongelmaan tekemällä laskelmat monimutkaisempien havaintojen pohjalta. Pelkän "teenkö vai en tee" -tarkistuksen sijaan ne voivat tarkistaa, kuinka paljon hahmo haluaa tai tarvitsee toiminnan suorittamista, antaen näiden huomioiden vaikuttaa toimintaan. Yleensä hyödynlaskentajärjestelmiä käytetään parantamaan siirtymälogiikkaa muissa tekoälyarkkitehtuureissa, mutta tekoälyn voi rakentaa myös täysin hyödynlaskennan pohjalta (Dawe ym. 2013, 54).

Caverna

Aiemminkin mainittu itse suunnittelemani Caverna on luolan managerointipeli ja reaaliaikainen strategiapeli. Pelissä erilaisille joukoille annetaan ohjeita, kuten “kaiva luolastoa” tai “treenaa lähitaistelua”, jolloin ohjeistettava yksikkö alkaa toteuttaa tätä käskyä.

Hahmoilla on kuitenkin erilaisia tarpeita, joita ne yrittävät saada tyydytetyksi toteuttaen samalla annettua tehtävää. Esimerkiksi hahmo saattaa tarvita ravitsemusta, jolloin se keskeyttää välillä annetun tehtävänsä käydäkseen syömässä. Joskus hahmolla voi olla tarve viihdyttää itseään, joten se lähtee huvitteluhuoneeseen pelailemaan.

Cavernassa ohjelmoijan tehtävänä on luoda hahmoista sen oloisia, kuin ne toimisivat oman tahtonsa ja ominaisuuksiensa ohjailemana, totellen kuitenkin samalla pelaajan antamia käskyjä.

Cavernan toteutus on vielä alkuvaiheissaan, mutta suunnitelmat ovat pitkällä. Pelissä tekoäly on monimutkainen, mutta sen ei tarvitse olla erityisen älykäs. Tämänhetkisen suunnitelman pohjalta valitsisinkin toteutuksessa hierarkkisen tilakoneen ja hyödynlaskentajärjestelmän hybridin.

Tekoälyn toiminta on jaettu hierarkkisesti sisäkkäisiin tiloihin ja tilakoneen mukaisesti tilojen väleillä on siirtymät. Näillä siirtymillä ei ole kuitenkaan samalla tavalla ehtoja, kuin perinteisessä tilakoneessa, vaan siirtymien haluttavuutta ja järkevyyttä lasketaan hyödynlaskentajärjestelmän tapaan. Esimerkiksi hahmon ollessa harjoittelemassa tarkistetaan kuinka paljon hahmon pitää jatkaa harjoittelua, kuinka paljon hahmon pitää käydä syömässä ja kuinka paljon hahmo haluaa rentoutua. Hyödynlaskentajärjestelmä pitää huolta, että hahmo pystyy painottamalla tekemään sitä, mikä on sillä hetkellä järkevintä tai halutuinta.

Erona Cavernassa käytetyn hybridi-mallin ja sumean tilakoneen välillä on tilojen äärellisyys. Cavernassa hahmo ei osaa tehdä useaa asiaa yhtä aikaa, joten hahmo on kerrallaan vain yhdessä tilassa.

3.5 Muita käytettyjä malleja

Tavoiteorientoituneet toimintasuunnittelijat (GOAP - Goal-Oriented Action Planners) toimivat lyhyesti kuvattuna siten, että tekoälylle annetaan valmis selitys siitä, kuinka maailma toimii, mutta kuitenkin mahdollisuus keksiä itse omat tapansa lähestyä sen ongelmia. Toisin sanoen tekoälylle annetaan lista mahdollisista toiminnoista ja niiden edellytyksistä sekä toiminnan vaikutuksista. Järjestelmä saa symbolisen esityksen maailman alkuasetelmasta, sekä tietyn määrän tavoitefaktoja, jotka sen tulisi saavuttaa. Toimintasuunnittelijan järjestelmä osaa niiden perusteella määritellä itselleen toimintojen ketjun, jonka perusteella kontrolloitava agentti alkaa muuttaa maailman tilaa sellaiseksi, että sen hetkisen tavoitteen fakta toteutuu. (Dawe ym. 2013, 55–57.)

Hierarkkiset tehtäväverkostot (HTN, Hierarchical Task Networks) ovat hieman tuntemattomampi tapa toteuttaa suunnittelijoita. Kuten muissakin suunnittelijoissa, esimerkiksi tavoiteorientoituneissa toimintasuunnittelijoissa, hierarkkiset tehtäväverkostot tähtäävät tiettyyn päämäärän toteuttavaan suunnitelmaan. Yhtenä erona useimpiin suunnittelijoihin, kuten GOAP, jotka toimivat *takaperin suunnittelulla* (backward planning), tehtäväverkostot toimivat *etuperin suunnittelemalla* (forward planning). (Dawe ym. 2013, 57–59.) Lisäksi Hierarkkiset tehtäväverkostot ajattelevat tehtäviä yksittäisten tehtävien sijaan hierarkkisessa verkostossa, abstraktimmalla tasolla (Wallace 2004, 230).

Takaperin suunnittelu tarkoittaa sitä, että suunnittelija aloittaa suunnittelun päämäärästä käyden toiminnat läpi takaperin. Esimerkiksi jos tekoäly haluaa oluen jääkaapista, on sen otettava olut jääkaapin sisältä. Jotta oluen saa jääkaapista, täytyy jääkaappi avata. Jotta jääkaapin saa avattua, täytyy jääkaapille mennä. Ja niin edelleen. Etuperin suunnittelu tarkoittaa sitä, että suunnittelija aloittaa maailman tämän hetkisestä tilasta ja suunnittelee toiminnat kyseisestä tilasta eteenpäin tähdäten haluttuun tilaan.

Hybridi-mallin tekoäly on yhdistelmä erilaisista tekoälystruktuureista. Hyvä hybridi sisältää hyviä puolia kaikista käytetyistä tekoälytyypeistä vähemmillä huonoilla puolilla.

3.6 Satunnaisuus ja painottaminen

Tekoälyyn saadaan lisättyä luonnollisuuden tuntua satunnaisuutta lisäämällä. Satunnaisuuden lisääminen voidaan kuitenkin toteuttaa useammalla eri tavalla. Perinteinen satunnaisuus perustuu suoriin todennäköisyyksiin, kuten kolikonheitossa klaavan ja kruunan todennäköisyydet 1/2 tai korttipakassa tietyn kortin nostamisen todennäköisyys 1/52. Suoran todennäköisyyden käyttäminen voi sopia joihinkin tapauksiin ja peleihin, mutta toisissa se ei olisi yhtä luonteva vaihtoehto. Muita sovellusmahdollisuuksia löytyy Gaussin jakaumasta (normaalijakaumasta) tai vaikkapa Perlin-kohinasta.

Gaussin jakaumalla (Gauss distribution) pystyy esimerkiksi simuloimaan osumatarkkuutta taistelusimulaatiossa. Mikäli tekoälyn ohjaaman ampujan osumatarkkuus toimisi tasaisella todennäköisyydellä, olisi maalitaulussa osumarypäs epärealistisen tasaisesti ympäri taulua. Osumatarkkuuden seuratessa normaalijakaumaa, tulee ryppäästä realistisen näköinen. (Rabin, Silva & Goldblatt 2013, 33.) Lisäksi ampujan ampumataitoa pystyy hyvin simuloimaan pienentämällä tai suurentamalla keskihajontaa.

Perlin-kohinaa (Perlin noise) käytetään yleensä luomaan visuaalisia efektejä, kuten esimerkiksi savua tai tulta. Rabinin ym. (2013, 38) mukaan Perlin-kohina voi myös auttaa simuloimaan muutoksia käyttäytymisessä ajan suhteen. Pelien tekoälyjä varten satunnaisuutekniikkana Perlin-kohina toimii hyvin siksi, että sen luoma satunnaisuus on niin kutsuttua koherenttia satunnaisuutta, missä satunnainen numerosarja ei heittelehdi niin voimakkaasti, vaan peräkkäiset numerot liittyvät toisiinsa (Rabin ym. 2013, 38).

Todellinen satunnaisuus ei usein vakuuta ihmispelaajaa. Ihmisen mielestä todellinen satunnaisuus ei vaikuta aidolta, sillä kun jokin on aidosti satunnaista, tiettyjä toistuvuuksia sattuu tapahtumaan. Silloin ihmismieli alkaa kehittää loogisia toistuvuuksia lyhyellä aikavälillä. Jos pelaaja epäonnistuu kolikonheittopelissä viisi kertaa peräkkäin, alkaa pelaaja helposti epäillä satunnaisuuden olevan pelille puolueellista. Sen vuoksi Rabinin ym. (2013, 34) mukaan todellista satunnaisuutta joudutaan välillä *filtteröimään* (filtered randomization), jotta liian pitkiä toistuvuuksia ei sattuisi. Filtteröinti tapahtuu siten, että mahdolliset satunnaisuudet arvotaan etukäteen, jonka jälkeen lista käydään läpi ja korjataan toistuvuuksien osalta (Rabin ym. 2013, 34).

Rabinin (2004, 75) mukaan esimerkiksi kokonaislukujen satunnaistamisen filtteröintisäännöt ovat seuraavat:

1. Rajoita lukujen toistamista.
2. Rajoita viimeisten kymmenen arvon aikana toistuvien lukujen määrää.
3. Rajoita nousevien ja laskevien sarjojen esiintymistä.
4. Rajoita peräkkäisiä satunnaistamisalueen ääripäiden lukuja. Esimerkiksi arvottaessa lukuja 0–9, sarjassa 9876987 on liikaa lukuja alueen yläpäästä.
5. Rajoita peräkkäisiä lukupareja, esimerkiksi 2244 tai 7711.
6. Rajoita välittömästi uudelleen esiintyvää lukukaksikkoa, kuten 3939 tai 7474.
7. Rajoita viimeisten kymmenen arvon aikana toistuvaa lukukolmikkoa, esimerkiksi 23578235 sisältää kahdesti sarjan 235.
8. Rajoita viimeisten kymmenen arvon aikana peilikuvana ilmestyvää lukukolmikkoa, esimerkiksi sitä, kuinka 634 esiintyy sarjassa 63481436.

Satunnaistamisen filtteröinnin tarpeellisuus riippuu pelin tyylistä. Esimerkiksi Cavernassa osassa hahmojen ominaisuuksien luomista käytetään Gaussin käyrää ja satunnaisesti luodut hahmot filtteröidään kevyesti, jotta pelaaja ei saa peräkkäin pelkästään huonoja sotilaita. Taistelussa tapahtuvaa satunnaisuutta (esim. lyönnin osumista) ei filtteröidä, koska sillä ei ole pelikokemuksen kannalta merkitystä. Golden Rollissa käytetään osaa satunnaistamisen filtteröinnin säännöistä tekoälyn luomien vuorojen järjestyksen satunnaistamisen aikana. Perlin-kohinaa käytetään Cavernassa hahmojen mielialan vaihtelun kertoimena. Tällöin hahmojen mielialat eivät vaihdu miten sattuu, vaan kehitys tapahtuu vähitellen.

Tietokoneella arvottu satunnaisuus ei ole koskaan todellista satunnaisuutta, vaan ainoastaan *pseudosatunnaisuutta* (mukasatunnaisuus, pseudo random). Tietokoneella laskettu satunnaisuus saa usein aikaan lievää painotusta tiettyihin arvoihin. On olemassa sellaista tietokonelaitteistoa, jolla pystyy luomaan todellisempaa satunnaisuutta kuin puhtaasti ohjelmistollisesti. Nämä laitteet voivat mitata muun muassa lämpökohinaa, valosähköistä kohinaa tai kosmista taustasäteilyä satunnaisuuden luomiseksi. Laitteistolla luotua satunnaisuutta ei nykyteknologialla pystytä ennustamaan, joten tällaista satunnaisuutta kutsutaan siksi joskus todelliseksi satunnaisuudeksi.

4 GOLDEN ROLL

Golden Roll on Greener Grass -nimisen yrityksen oma mobiilipeliprojekti. Peli seuraa tunnetun Yatzy-pelin sääntöjä pienillä eroavaisuuksilla ja uhkapelimäisillä lisäyksillä. Alun perin pääsin Golden Roll -projektista osalliseksi harjoittelijan roolissa Greener Grassilla, minkä jälkeen olen toteuttanut peliä koko vuoden 2017 ajan. Opinnäytetyön aiheeksi valitsin tekoälyn erityisesti pelin tekoälyvastustajan *Haasteet*-ominaisuutta varten. Aihevalintaani olen perustellut aiemmin Johdanto-osiossa.

4.1 Pelin säännöt ja ominaisuudet

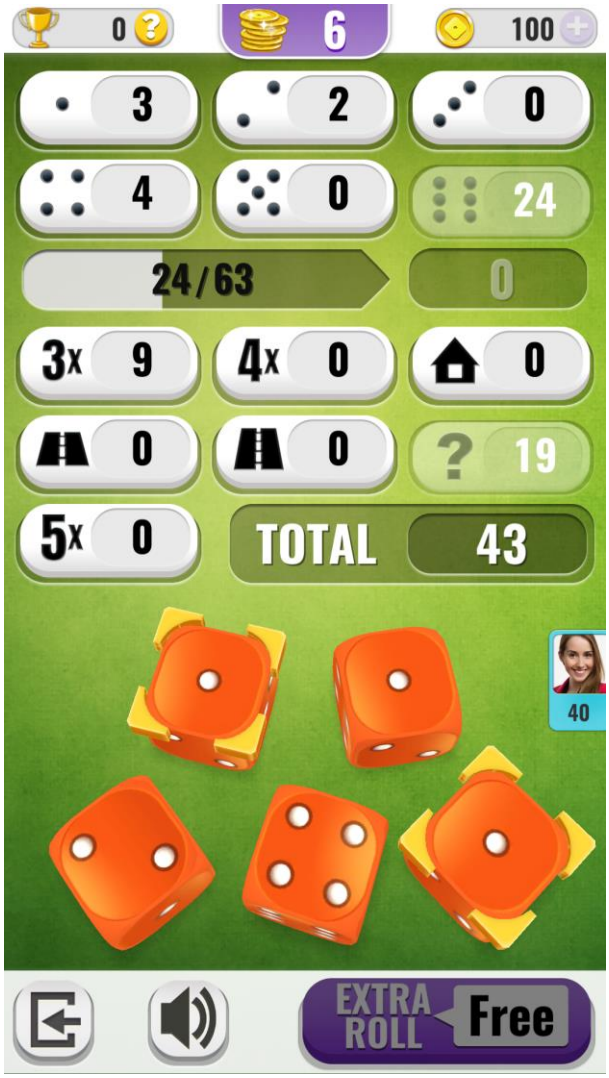
Esittelen ensin Golden Rollin säännöt ja ominaisuudet. Perinteisen Yatzy-pelin sääntöjen tuntemisesta on hyötyä Golden Rollin ymmärtämisessä, vaikka Golden Roll eroaakin Yatzysta muutamilta osin.

Ensinnäkin pelin valuuttana toimivat kolikot, joita pelaaja saa alkupankkiinsa pienen määrän. Kolikoita saa lisää voittamalla *haasteita* (challenge), voittamalla verkossa muita pelaajia, sekä ostamalla. Verkkopelin aloittaminen maksaa *pöytämaksun* (ante), mikä lisätään pelin *pottiin* (pot). Pöytämaksusta menee *talon leikkaus* (rake), ennen pottiin lisäystä. Kuvan 1 yläreunan oikealla näkyy pelaajan kolikot, ja yläreunassa keskellä potissa tällä hetkellä olevat kolikot.

Itse matsissa pelaajalla on käytössään viisi noppaa, ja jokaisella vuorolla kolme heittoa kaikilla viidellä nopalla. Heittojen välissä pelaaja valitsee, mitkä nopista lukitaan eli *pidetään* (keep) edellisen silmäluvun osoittamassa arvossa, eli jätetään heittämättä. Pelaaja voi tietenkin myös heittää uudestaan kaikkia noppiaan. Kuvassa 1 kaksi noppaa silmäluvuilla yksi on lukittuna. Alareunan “Extra Roll” -nappi tummentuneena tarkoittaa, että kaikki on heitot käytetty, myös *lisäheitto* eli Golden Roll. Kuva 1 on ajalta, kun lisäheiton nimi oli väliaikaisesti Extra Roll. Jokaisella vuorolla pelaajalla on käytössään yksi Golden Roll, joka maksaa kolikoita. Näistä kolikoista otetaan ensin leikkaus ja loput lisätään pottiin.

Kun pelaaja päättää lopettaa vuoronsa, on hänen valittava jokin kolmestatoista pisteytyskategoriasta, johon heitto merkitään ylös. Yhteensä kolmeentoista (13) kategoriaan sisältyvät numerokategoriat 1-6 sekä lisäksi *kolme samaa*, *neljä samaa*, *täyskäsi*, *lyhyt*

suora, *pitkä suora*, *sattuma* ja *Viisi samaa*. Kuvassa 1 kategorioista on aiemmilla vuoroilla käytetty numerokategoria 6 sekä sattuma. Nykyisillä nopilla saatavat mahdolliset pisteet näkyvät käyttämättömissä kategorioissa.



KUVA 1. Pelinäkymä (lokakuu 2017).

Numerokategorioissa pisteitä saa niistä nopista, joiden silmäluku on sama kuin kategorian numero. Esimerkiksi: jos kolme nopista on kakkosia, ja kaksi jotain muuta, kategoriasta kaksi saisi 2x3 eli yhteensä 6 pistettä. Numerokategorioihin liittyy lisäsääntö *Upper Bonus*. Upper Bonuksen eli yhteensä 35 pistettä saa, kun numerokategorioista on pisteytettynä 63 pistettä tai enemmän.

Kolme samaa ja neljä samaa pystyy pisteyttämään, kun nopissa on vähintää kolme tai neljä samaa silmälukua. Pisteitä saa silti kaikki noppien silmäluvut yhteenlaskettuna.

Täyskäsi on kolme samaa ja pari ja täyskäsi on yhteensä 25 pisteen arvoinen. Pieni suora on neljä peräkkäistä lukua, 30 pistettä, ja pitkä suora on viisi peräkkäistä lukua, 40 pistettä. Sattuma-kohtaan voi laittaa mitä tahansa, minkä takia joissain Yatzyn versioissa sitä on myös kutsuttu *pelastusrenkaaksi*.

Viisi samaa on nimensä mukaisesti viisi mitä tahansa samaa nopan silmälukua. Viisi samaa on aina 50 pisteen arvoinen heitto, ja mikäli kategoria on jo pisteytetty, voi heiton pisteyttää minne tahansa. Tämän säännön nimi on *Ylimääräinen viisi samaa* (Additional Five of a Kind) ja siitä saa pisteitä sen, mitä saisi kyseisillä noppien silmäluvuilla normaalisti pisteytettävästä kategoriasta, sekä lisäksi Viisi samaa -säännön antamat 50 pistettä.

Pelin voittaa se pelaaja, jolla on pelin loputtua eniten pisteitä. Voittaja saa kaikki potissa olevat kolikot itselleen. Tasapelitilanteessa kolikot jaetaan tasapelin saaneiden pelaajien kesken.

4.2 Ensimmäisestä tekoälystä kohti haasteet-osion vaatimaa tekoälyä

Tarve yksinkertaiselle tekoälylle tuli heti projektin alkuvaiheessa pelin testaamista varten. Ensimmäinen tekoäly heitti aina vuoronsa alussa kerran noppia, ja tämän jälkeen kävi läpi kaikki kategoriat valiten niistä sen, mistä saa suurimmat pisteet tällä yhdellä heitolla. Tekoäly ei siis ollu millään tavalla älykäs, vaan toimi lähes täysin sattumanvaraisesti.

Peliin alettiin pian suunnitella Haasteet-ominaisuutta. Tällöin vaativimmaksi muodostui suunnitella sellainen toimiva tekoäly, joka toimisi haasteiden vaatimien ominaisuuksien puitteissa. Golden Rollissa Haasteiden idea on antaa pelaajalle tekoälyvastustaja, joka paranee portaittain. Ensimmäisen vastustajan voittaa todennäköisesti jokainen pelaaja ensimmäisellä yrityksellä, mutta haasteen viimeinen vastustaja on jo niin vaikea, että se vaatii useamman yrityksen jopa parhaimmaltakin pelaajalta. Jokaisesta portaasta saa kolikoita ja haasteen lopussa mahdollisesti joitain muitakin palkintoja.

Jokaisen portaan läpäisypistemäärä on ennalta määrätty jokaiselle haasteelle, joten tekoälyn tulee osata pelata niin, että jokaisella kerralla tekoäly saisi tuon tarvittavan pistemäärän. Toisaalta olisi hyvä, ettei tekoäly pelaisi joka kerralla liian samannäköistä peliä.

4.3 Suunnitelma

Tekoälyn suunnittelun lähtökohtina olivat kaksi vaatimusta. Ensinnäkin tekoälyn tulisi saada aina se pistemäärä, mikä on sille määrätty, sekä toiseksi, tekoälyn tulisi pelata mahdollisimman paljon eri tavoilla. Tekoälyn olisi hyvä olla edes jollakin tasolla orgaanisen eli luonnollisen tuntuinen.

Ensimmäisissä suunnitelmissa lähdettiin liikkeelle tekoälystä, jolle heitot arvottaisiin samalla tavalla kuin pelaajalle, ja tekoäly valitsisi pelaamisen aikana pistensä niin, että lopputulos olisi toivottu. Ennen kuin tekoälyä alettiin toteuttaa, huomattiin, että tämänkaltaisen tekoälyn epävarmuustekijät ovat suuret. Noppien silmäluvut voivat olla sattumalta ihan mitä tahansa, jolloin tekoälyn on vaikea ennakoida, mihin kategoriaan luvut sijoitettaisiin.

Siksi olikin järkevintä lähteä siitä, että määrätty pistemäärä jaetaan pistekategorioiden kesken, ja sitten valitaan niiden järjestys ja nopat. Tekoälytyypiksi valittiin sääntöperusteinen skriptattu tekoäly. Tekoäly skriptaa sille annettujen sääntöjen mukaan etukäteen omat vuoronsa, jotka se sitten toteuttaa järjestyksessä. Luokkaa varten luotiin kuviossa 4 näkyvät rajapinnat *ITurnList* ja *ITurn*.

```
public interface ITurn {
        int Score { get; }
        int Rule { get; }
        int[] Dice { get; }
}
public class Turn : ITurn { ... }
public interface ITurnList {
        void Init(string opponentPlayerId, int targetScore);
        void Init(string opponentPlayerId, int targetScore, int errorMargin);
        ITurn GetTurn(int index);
}
public class FixedScoreTurnGenerator : ITurnList { ... }
```

KUVIO 4. ITurn- ja ITurnList-rajapinnat.

ITurnList-rajapinnalla on käytännössä kaksi funktiota: vuorojen alustaminen ja generointi *Init*-metodilla ja *GetTurn*, jolla pystyy kysymään indeksiä kohden vuoron. ITurn-rajapinnalta saa vuorolla tavoitellun pistemäärän, kategorian, joka vuorolla pisteytetään, sekä noppien silmäluvut, jotka vuorolla heitetään.

4.4 Toteutus ja testaus

Kuten edellä kuvattiin, tekoäly toteutettiin lopulta sääntöpohjaisena tekoälyn esiskriptataamilla vuoroilla, koska suunnitteluvaiheessa päädyttiin siihen, että tekoälyn on parasta olla sopivan ennalta-arvattava, jotta tietty pistemäärä pystyttäisiin saamaan. Kun tekoälyn vuoro alkaa, hakee pelaustekoäly vuoronsa ITurnList-rajapinnan toteuttavalta luokalta. Vuoro sisältää nopan silmäluvut, sekä pisteytettävän kategorian. Noppien tulevat silmäluvut asetetaan paikalleen ja tekoäly heittää noppia kerran. Tämän jälkeen tekoäly valitsee vuorossa määrätyn kategorian ja lopettaa vuoronsa.

Vuoroja varten *FixedScoreTurnGenerator*-luokka luo pisteytyksen tietyssä järjestyksessä. Ensin pisteytetään kaikki vakiopisteelliset kategoriat, sitten loput alaosan pisteet ja lopuksi yläosan pisteet. Jokaista osaa varten on määritetty tietyt vakiot, joihin osioita ja niiden kategorioiden pisteitä verrataan. Jokaisella kategorioiden kategorialla on maksimipisteet, jotka osioon voi kohdistaa ja esimerkiksi vakiopisteellisissä kategorioissa todennäköisyys pisteyttää perustuu suoraan siihen, kuinka suuren osan näistä maksimipisteistä kyseiset pisteet veisivät.

Kuviosta 5 nähdään, kuinka toteutusta testatessa havaittiin huomattavasti virheitä pisteytettäessä kategorioita suuremmilla pisteillä. Testejä tehtiin yhteensä tuhat kappaletta 75 pisteen välein. Kuvion riveillä ovat järjestyksessä valmistumisajankohta, pistemäärä, johon generointi tähtäsi, onnistuneiden generointien lukumäärä, ne kerrat, kun pisteytettiin liikaa, kerrat, kun pisteytettiin liian vähän, ja uudella rivillä prosentuaalinen onnistuminen.

KUVIO 5. Ensimmäisiä testejä

Ensimmäisen testausvaiheen jälkeen päätettiin, että vuorojen luomistekoälyn olisi syytä olla modulaarinen eli osista koostuva. Koska kategoriat oli valmiiksi järjesteltynä kolmeen eri osioon (alaosan vakiopisteelliset kategoriat, alaosan muut kategoriat, numerokategoriat), pystyttiin kategoriat jakamaan tietyillä muutoksilla omiksi metodeikseen ja optimoimaan tarvittavat muuttujat niin, että testaaminen onnistuu myös erikseen.

Koska numerokategoriat generoidaan viimeisenä, niiden generoinnin alussa lisätään viisi samaa -heittoja tarpeeksi, jotta pisteytys on mahdollinen. Kuvion 6 alussa tarkistetaan, onko Viisi samaa pisteytetty laisinkaan, ja mikäli on, aletaan niitä lisäämään numerokategorioihin lisää. Koodissa on legacy-syistä käytetty muuttujien niminä sanaa “Yatzy” pelissä olevan kategoriannimen “Viisi samaa” sijaan.

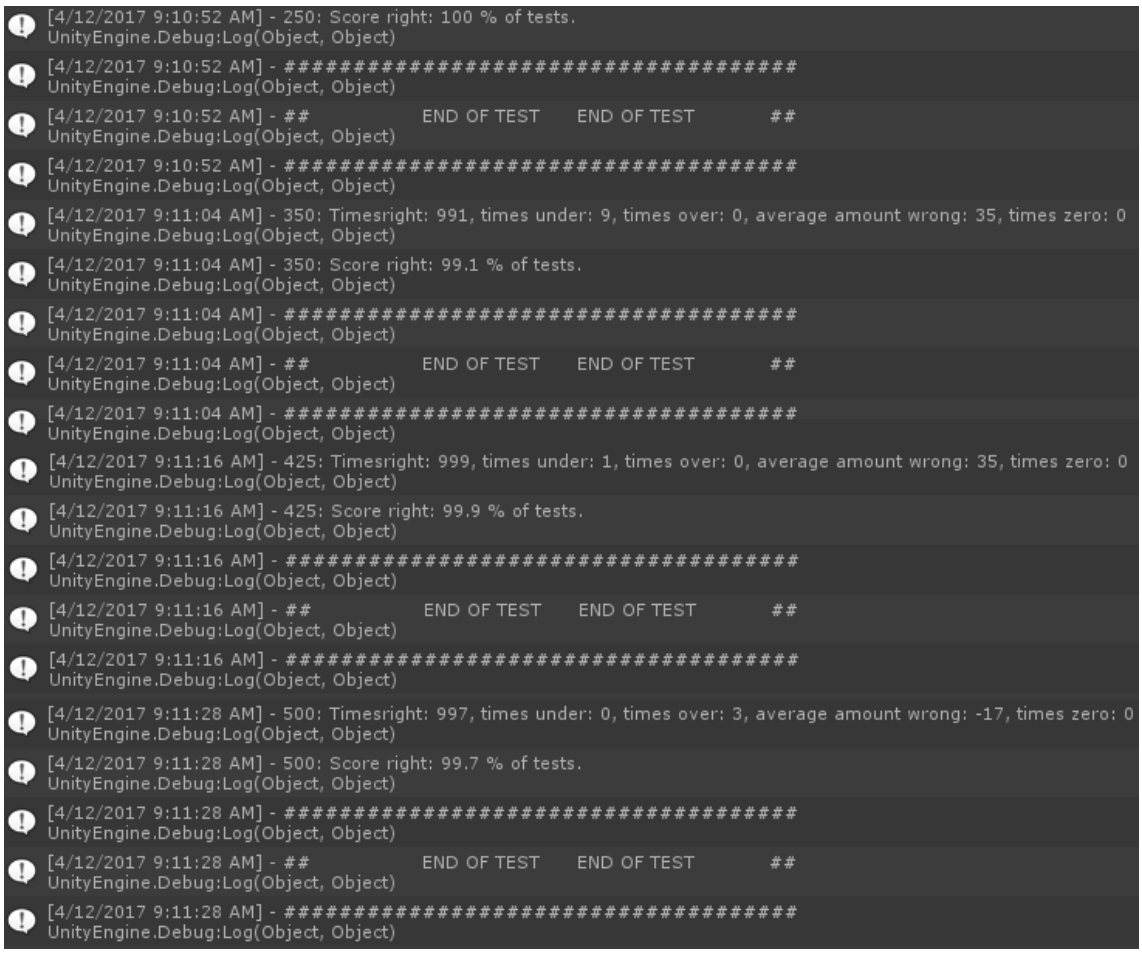
```
while (yatzyCount > 0 && scoreLeft > maxScoreWithoutYatzies) {
                int min = Mathf.Min(1, numberScoresNotMaxed.Count - 1);
                // exclude number 1 aslap
                int max = Mathf.Max(1, numberScoresNotMaxed.Count - 1, min + 1);
                //exclude number 6 aslap
                int index = randomizer.Range(min, max);

                NumberScore numberScoreRule = null;
                int score = 0;
                do {
                                numberScoreRule = numberScoresNotMaxed[index];
                                score = (numberScoreRule.Number * 5
                                                + additionalYatzyScore);
                                index--;
                } while (score > scoreLeft && index >= 0);

                scoresToGet[numberScoreRule] += score;
                scoreLeft -= score;
                maxScoreWithoutYatzies -= numberScoreRule.Number * 4;
                numberScoresNotMaxed.Remove(numberScoreRule);
                ++yatzyCount;
}
```

KUVIO 6. Esimerkkikoodia numerokategorioiden pisteiden generoinnista.

Tekemällä yksittäisiä testejä esimerkiksi numerokategorioille pystyttiin löytämään suurimmat ongelmakohdat ja optimoimaan vakioiden ja raja-arvojen arvoja, sekä lisäämään uusia tarkistuksia. Bugien korjaamisen ja vakioiden parantelun jälkeen testeistä saatiin huomattavasti parempia tuloksia, kuten jo kuviosta 7 nähdään. Tässä testien versiossa jokaisen testin välissä oli nuo suuret ylimääräiset tulostukset ”END OF TEST” ja #-merkkirivit, koska testejä tehtiin huomattavasti enemmän kerralla. Kuviossa näkyy pistemäärä 350, joka on vaikein saada oikein. Siinä onnistuttiin 99,1 prosentissa yrityksistä. Koska virhemarginaali oli alle prosentin, versio uskallettiin lisätä testattavaan peliin.



KUVIO 7. Testejä korjausten jälkeen.

Kuviosta 7 testien 350 ja 425 riveiltä huomataan, että virheen määrä virhetilanteessa oli tällä versiolla keskimäärin 35. Kyseessä on sama pistemäärä, kuin Upper Bonuksesta saatava, mikä ei voi olla sattumaa. Virhe merkitsee sitä, että pisteisiin oli laskettu Upper Bonus, mutta todellisuudessa yläosan pisteytykseen oli generaatiossa luotu liian matalat pisteet.

Kun kategorioiden generointi oli saatu optimoitua alle prosentin virhemarginaaliin, toteutettiin noppien generointi vuoroille. Noppien generoinnin testaamista varten noppageneraattori irrotettiin omaksi luokakseen, minkä lisäksi sekin jaettiin modulaarisesti, jotta jokaista kategoriaa voi testata erikseen.

Noppia generoidessa tulee ottaa huomioon kategorioiden pisteytysjärjestys, jotta tekoäly ei vaikuttaisi liian yksinkertaiselta. Lisäksi, mikäli generointi luo viisi samaa noppaa johonkin kategoriaan kuitenkaan pisteyttämättä Ylimääräistä viittä samaa, täytyy kyseinen pisteytys tehdä ennen kuin Viisi samaa pisteytetään. Tämän lisäksi alhaisilla pis-

teillä Viisi samaa -kategoria on usein 0, joten se täytyy pisteyttää loppuvaiheessa, todennäköisimmin viimeisenä, kuten pelaajakin pisteytyksen tekisi.

Noppien arvonnan ja asettelun jälkeen arvotaan vuorojärjestys painotetusti. Vuoroille on generaatiossa annettu tiettyjä painokertoimia, joita on neljä tasoa. Esimerkiksi nollaksi pisteytetty Viisi samaa on kaikista alhaisimman painokertoimen vuoro, jolloin se todennäköisesti päätyy vuorolistan viimeiseksi. Mikäli taas tekoälylle on generoitunut useampi Viisi samaa, on Viisi samaa -kategorian vuoro korkeinta painokerrointa, jotta se varmasti pisteytetään ennen Ylimääräinen viisi samaa -tapauksia.

4.5 Tutoriaalivastustaja

Tekoälyn modulaarisesta toteutustavasta johtuen tekoäly oli helppo laajentaa tutoriaalivastustajaksi. Tutoriaalivastustajan tapauksessa kategorioita ja pisteitä ei esiskriptata tekoälyllä, vaan pisteet ja kategoriat annetaan tekoälylle ohjelmoijan editorissa tekemänä listana. Tutoriaalin vapaan pelaamisen aikana tekoäly tarkastaa pelaajan pisteistä, ettei se jää liikaa jälkeen tai vastaavasti mene liikaa edelle. Sen ansiosta viimeisellä vuorolla tekoäly johtaa väistämättä, jolloin pelaajalle saadaan opetettua Golden Rollin tekeminen, sekä luotua hiukan jännityksen ja draaman tuntua.

Kun pelaaja on pisteyttänyt vuoronsa, tarkistaa tekoäly sen hetkisten kokonaispisteiden eron. Mikäli piste-ero on yli tietyn raja-arvon kumpaan tahansa suuntaan, muuttaa tekoäly pisteitä tarpeen mukaan, kuitenkin pitäen pisteet laillisina vuoroa varten varattuun kategoriaan. Tarkistuksen jälkeen tekoäly määrittelee näihin pisteisiin ja kategoriaan sopivat nopat.

4.6 Yhteenveto

Haasteet-ominaisuuden tekoäly on käytännössä kaksiosainen. Siihen sisältyvät vuorogeneroinnin tekoäly ja peliä näillä vuoroilla pelaava tekoäly. Sääntöpohjainen vuorogenerointitekoäly käy läpi kaikki kategoriat hieman satunnaistetussa järjestyksessä ja arpoo näihin pisteet tietyillä tarkistuksilla. Sama tekoäly tekee generoinnin jälkeen tarkistuksen, jotta pisteytys on mennyt oikein. Seuraavaksi generoidaan kategorioiden pisteisiin sopivat nopat sopivasti satunnaistamalla. Lopuksi tekoälyn generoimat vuorot arvotaan pelattavaan järjestykseen sopivilla painotuksilla.

Vuorogeneroinnin tekoäly tekee oman työnsä ennen itse matsin alkamista. Kun matsi alkaa, pääsee vuoroon pelaava tekoäly. Haasteet-osion pelaava tekoäly on lopulta hyvin yksinkertainen. Kun pelilogiikka lähettää kaikille pelaajille viestin vuoron alkamisesta, tekoäly hakee kyseisen vuoron vuorolistalta. Se asettaa noppien tuleviksi arvoiksi vuorolle merkityt silmäluvut ja heittää noppia. Kun nopat on heitetty, valitsee tekoäly vuorolle merkityn kategorian lopettaen vuoronsa.

Tutoriaalin tekoäly käyttää samaa vuorolistan rajapintaa vuorojen hakemiseen, mutta sen vuorot on esiskriptattu ohjelmoijan toimesta tekoälyn sijaan. Matsin aikana pelaava tekoäly aloittaa oman vuoronsa vasta, kun pelaaja on pelannut oman vuoronsa. Tekoäly ei seuraa vuorolistaa täysin orjallisesti, vaan vertaa omia ja pelaajan pisteitä toisiinsa varmistaakseen, ettei putoa liikaa pisteissä taakse tai mene liikaa edelle.

Haastetekoäly antaa pelille lisää sisältöä mahdollistaen portaittain vaikeutuvan haasteosion peliin. Sen tietty satunnaisuus tekee pelaamisesta mielenkiintoisempaa, kuin joka kerta tismalleen samalla tavalla pelaava tekoäly tekisi. Tutoriaalitekoäly tekee tutoriaalista erilaisen eritasoisille pelaajille mahdollistaen kuitenkin pelaajalle viimeisen vuoron loppukirin.

5 POHDINTA

Tekoälyn toteuttaminen peliprojektissa vaatii huolellista suunnittelua. Jotta peliin voi alkaa toteuttaa tekoälyä, on hyvä tietää etukäteen, mitä tekoälyltä halutaan. Hyvä suunnittelu helpottaa tekoälyn toteuttamista ja antaa mahdollisuudet sen laajentamiseen myöhemmin. Muiden pelien tekoälyjen tutkiminen auttaa hyvän ja tarpeenmukaisesti toimivan tekoälyrakenteen valitsemisessa, jolloin tekoälyn toteutus pääsee alkuun hyvällä vauhdilla.

Golden Roll -projektissa Haasteet-osion tekoälyn toteutuksessa päädyttiin voimakkaammin skriptattuun tekoälyyn autonomisemman sijaan, jotta tietynlainen ennaltaarvattavuus ja -määriteltävyys säilyisi. Autonomisemmalla tekoälyllä olisi voitu saada aikaan orgaanisemman tuntuinen vastustaja, mutta turhan autonominen tekoäly on arvaamaton, jolloin lopputulos ei sittenkään olisi ollut toivotun kaltainen. Kaikista orgaanisimman tuntuinen tekoäly heittäisi pelaajan tavoin kolme yritystä noppia ja pisteyttäisi niiden mukaan. Tällöin tekoälyn saama lopullinen pistemäärä ei kuitenkaan välttämättä olisi tarkoituksenmukainen.

Orgaanisuuden tuntua saatiin lisättyä satunnaistamalla pisteytysjärjestystä sekä satunnaistamalla yksittäisistä kategorioista saatuja pistemääriä. Siksi tekoälyllinen esiskriptaus oli todennäköisesti paras toteutusvaihtoehto. Sääntöpohjainen tekoäly on tarpeeksi yksinkertainen, että sillä on helppo ja suoraviivainen toteuttaa vuorojen tekoälyllinen generointi. Tavoiteorientoituneella toimintasuunnittelijalla olisi pystynyt toteuttamaan erittäin orgaanisen ja luotettavan vuorogeneraation, mutta tekoälyn tuotantokustannukset olisivat nousseet liian suuriksi eikä hyöty olisi ollut merkittävän suuri valittuun malliin nähden.

Haasteellisin vaihe tekoälyn toteuttamisessa oli tekoälyn optimointi niin, että jo yhdellä yrityksellä tekoäly saisi generoitua vuorolistan. Kaikkien vakioiden ja muuttujien säätäminen kohdilleen sekä uusien vakioiden ja muuttujien asettaminen vei aikaa ja vaati suunnittelua työn ohessa. Alkuperäisen toteutuksen kanssa testaaminen oli haasteellista, mutta tämä ratkaistiin hajottamalla generaatiota osiin. Modulaarisessa toteutuksessa hyvä puoli oli se, että arvojen muuttamisen jälkeen pystyi testaamaan erikseen jokaista osiota. Silloin ongelmien kertymäkohdat tulivat näkyviksi.

Soft launch -vaiheessa pelaajamäärät jäävät usein alhaisiksi. Pelaajamäärien ollessa alhaiset täysin pelaajavetoinen peli tuntuu autiolta ja tekeminen saattaa loppua pelaajalta kesken. Tekoälyvastustaja tuo lisää sisältöä ja tekemistä peliin. Golden Rollissa tekoälyvastustajaa käytetään tietyllä tavalla tulonlähteenä pelaajille. Voittamalla haastevastustajia pelaajat saavat kolikoita pelata muita pelaajia vastaan, silloin kun muita pelaajia löytyy.

Jotta Golden Rollin tekoälystä saadaan miellyttävä vastustaja, kehitystyötä on hyvä jatkaa vielä pienellä työpanoksella. Työstämällä vielä noppien generointia vuoroihin ja vuorojen järjestyksen painottamista, tekoälyä voi todennäköisesti käyttää pelin muissakin tulevissa sisällöissä.

Erilaisissa peleissä tarpeet tekoälyn suhteen ovat hyvin erilaiset. Jopa saman peligenren sisällä tarpeet voivat vaihdella huomattavasti. Jos peli on moninpeli, tekoälyä ei välttämättä tarvita juuri lainkaan, kun taas yksinpelikampanjassa tai reaaliaikastrategiapelin pikapelissä tekoäly voi olla korvaamaton. Millaiseksi tekoälyn tyyppi, monimutkaisuus ja älykkyys kannattaa valita, riippuu paljon projektin luonteesta, mutta valinta voi riippua täysin myös pelin tekijöiden preferensseistä.

LÄHTEET

Buckland, M. 2005. Programming Game AI by Example. Sudbury, MA, USA. Wordware Publishing

Dawe, M., Gargolinski, S., Dicken, L., Humphreys, T. & Mark, D. 2013. Behavior Selection Algorithms: An Overview. Teoksessa Rabin, S. (toim.) Game AI Pro: Wisdom of Game AI Professionals. Boca Raton, FL, USA. Taylor and Francis Group

Dill, K. 2013. What is Game AI? Teoksessa Rabin, S. (toim.) Game AI Pro: Wisdom of Game AI Professionals. Boca Raton, FL, USA. Taylor and Francis Group

Fu, D. & Houlette, R.. 2004. The Ultimate Guide to FSMs in Games. Teoksessa Rabin, S. (toim.) AI Game Programming Wisdom 2. Hingham, MA, USA. Charles River Media

Meyer R. 2017. Tree's Company: Systemic AI Design in 'Just Cause 3'. Avalanche Studios. http://gdcvault.com/play/1024605/Tree-s-Company-Systemic-AI

Poole, D., Mackworth, A., & Goebel, R. 1998. Computational intelligence: A logical approach. NY, USA. Oxford University Press

Rabin, S. 2004. AI Game Programming Wisdom 2. Hingham, MA, USA. Charles River Media

Rabin, S. 2013. Game AI Pro: Wisdom of Game AI Professionals. Boca Raton, FL, USA. Taylor and Francis Group

Rabin, S., Goldblatt, J. & Silva, F. 2013. Advanced Randomness Techniques for Game AI: Gaussian Randomness, Filtered Randomness, and Perlin Noise. Teoksessa Rabin, S. (toim.) Game AI Pro: Wisdom of Game AI Professionals. Boca Raton, FL, USA. Taylor and Francis Group

Russel, S. & Norvig, P. 2010. Artificial Intelligence: A Modern Approach, Third Edition. Upper Saddle River, NJ, USA. Prentice Hall

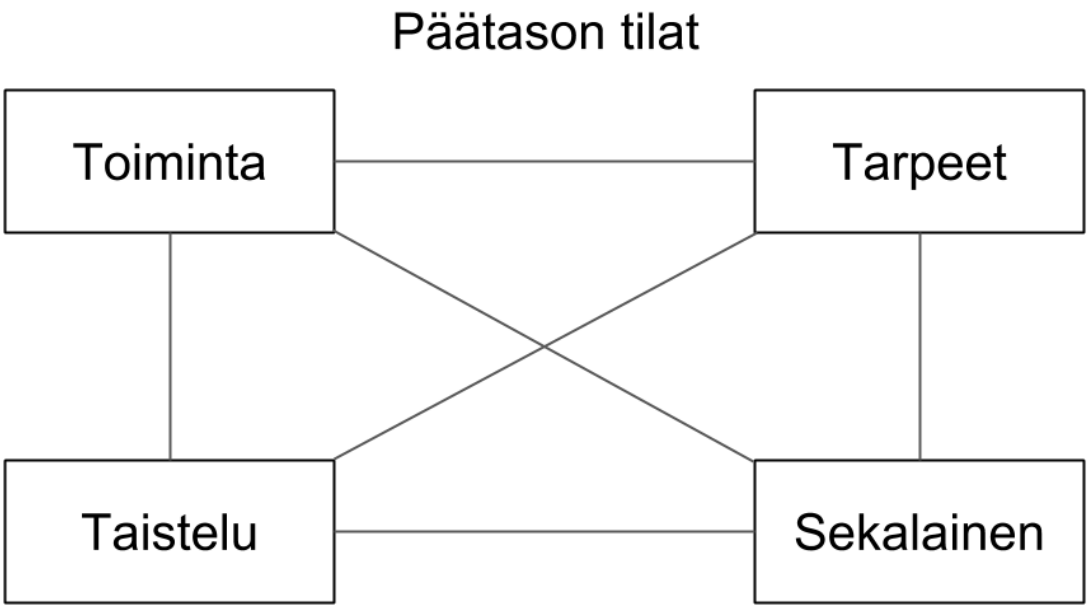
Schwab B., 2004. AI Game Engine Programming. Hingham, MA, USA. Charles River Media

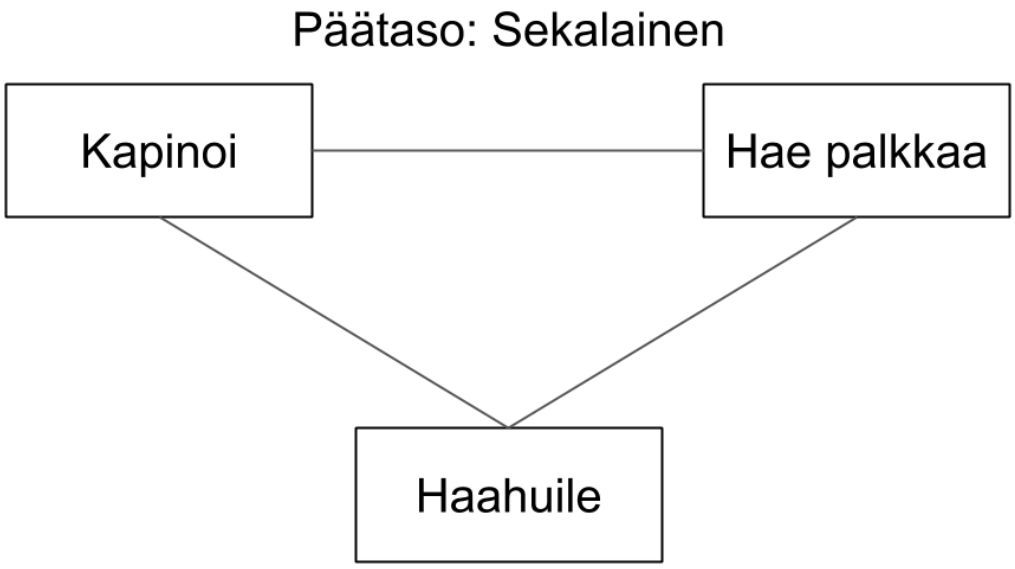
Skoog K. 2017. theHunter: Call of the Wild – Designing Believable, Simulated Animal AI. http://www.karineskoog.com/thehunter-call-of-the-wild-designing-believable-simulated-animal-ai/

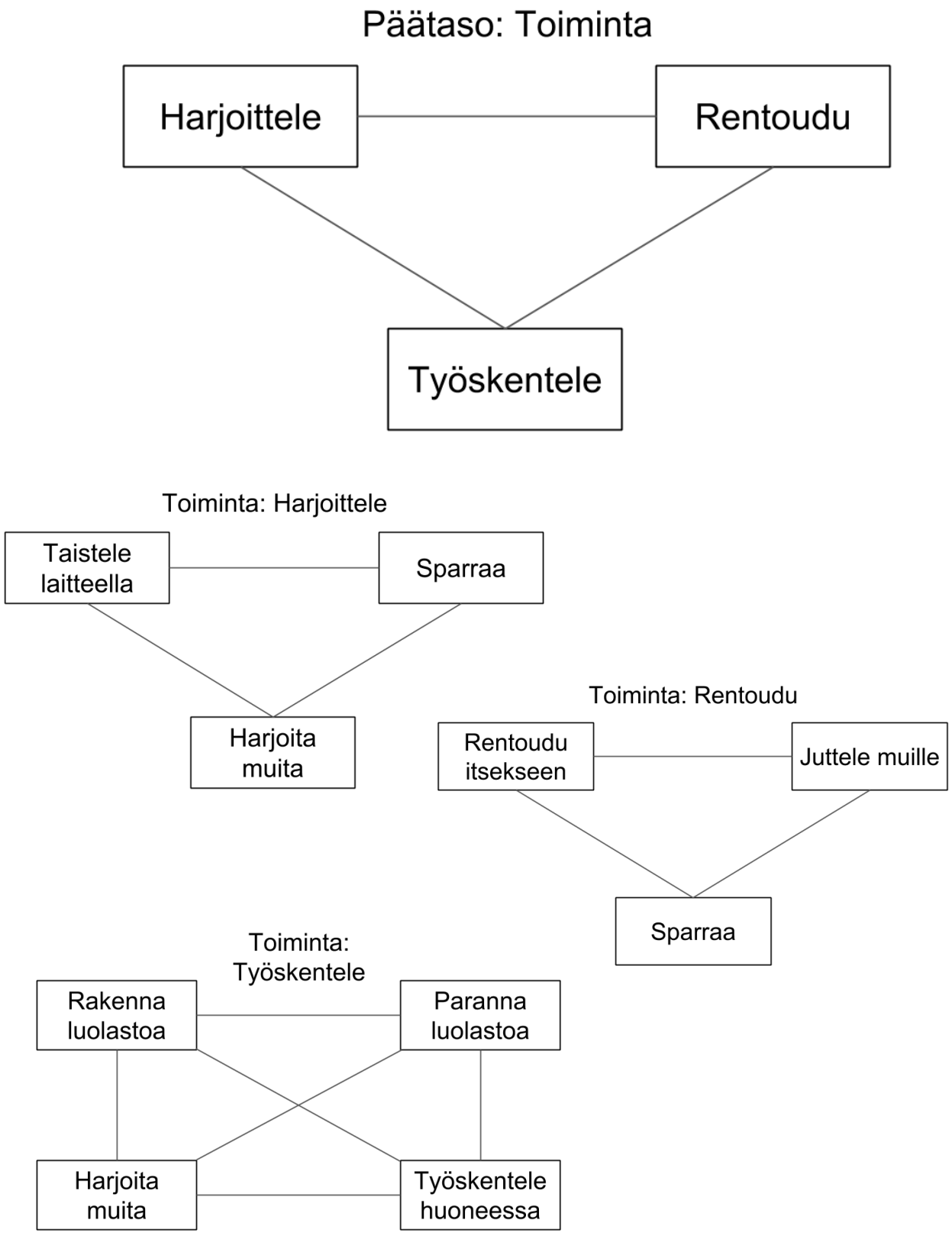
Wallace, N. 2004. Hierarchical Planning in Dynamic Worlds. Teoksessa Rabin, S. (toim.) AI Game Programming Wisdom 2. Hingham, MA, USA. Charles River Media

LIITTEET

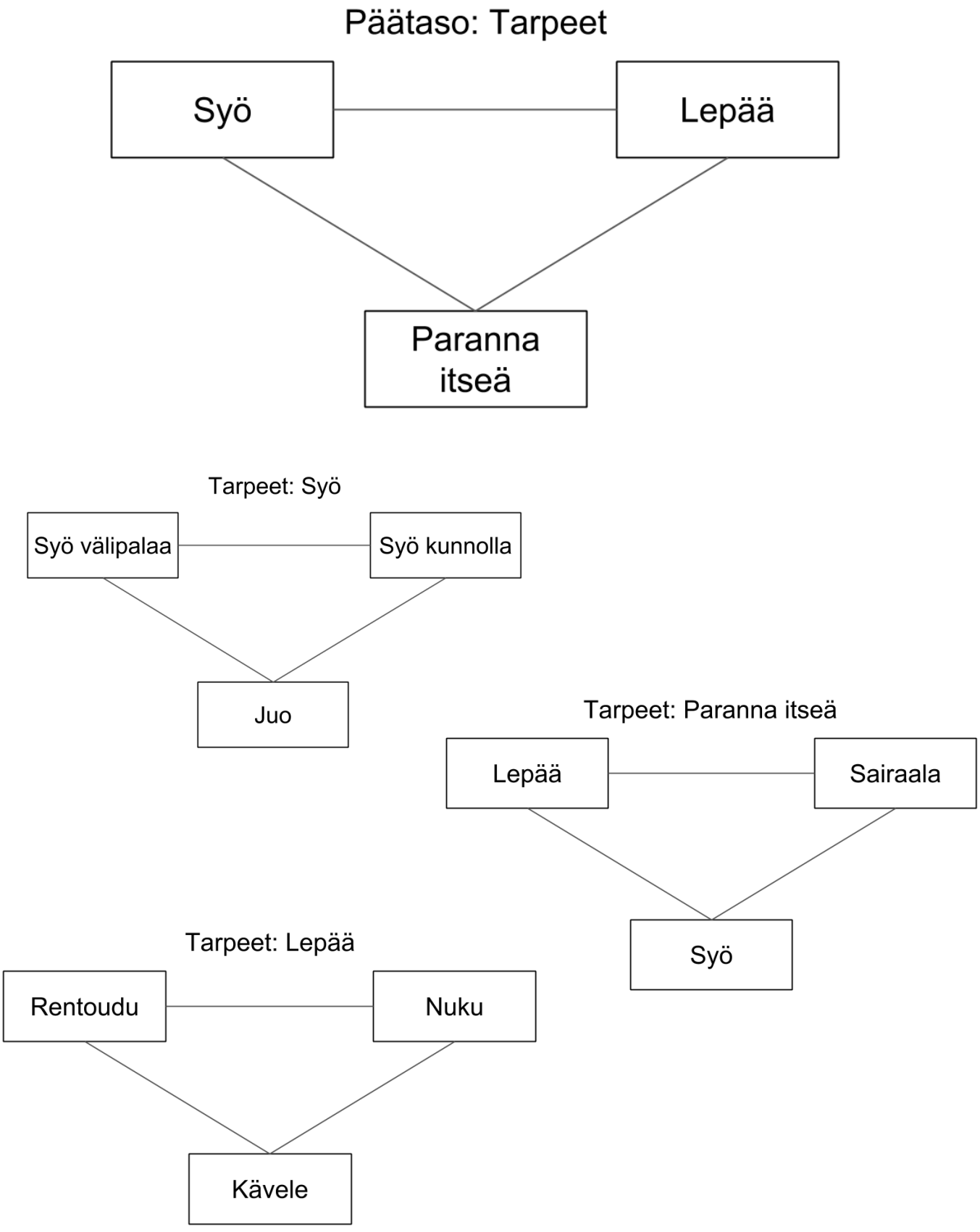
Liite 1. Caverna: Tekoälyn tilakoneen rakennesuunnitelmaa 1 (4)



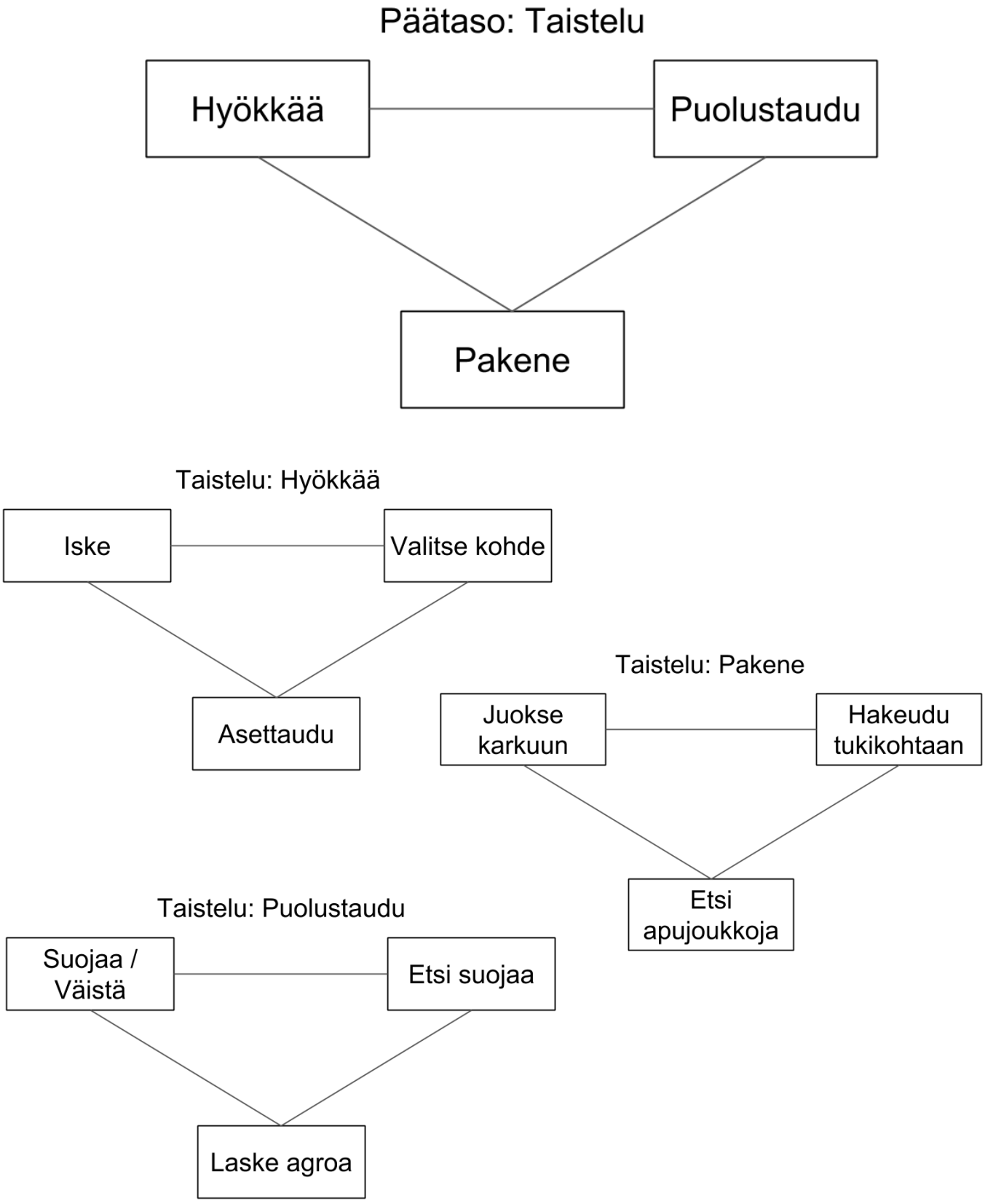



Päätaso: Toiminta
Harjoittele
Rentoudu
Työskentele
Toiminta: Harjoittele
Taistele laitteella
Sparraa
Harjoita muita
Toiminta: Rentoudu
Rentoudu itsekseen
Juttele muille
Sparraa
Toiminta: Työskentele
Rakenna luolastoa
Paranna luolastoa
Harjoita muita
Työskentele huoneessa

3(4)


Päätaso: Tarpeet
Syö
Lepää
Paranna itseä
Tarpeet: Syö
Syö välipalaa
Syö kunnolla
Juo
Tarpeet: Paranna itseä
Lepää
Sairaala
Syö
Tarpeet: Lepää
Rentoudu
Nuku
Kävele

4 (4)


Päätaso: Taistelu
Hyökkää
Puolustaudu
Pakene
Taistelu: Hyökkää
Iske
Valitse kohde
Asettaudu
Taistelu: Pakene
Juokse karkuun
Hakeudu tukikohtaan
Etsi apujoukkoja
Taistelu: Puolustaudu
Suojaa / Väistä
Etsi suojaa
Laske agroa